Operating Systems And Applications For Embedded Systems

Profiling and Tracing

Plan

Profling with top

top is a simple tool that doesn't require any special kernel options or symbol tables. There is a basic version in BusyBox, and a more functional version in the procps package which is available in the Yocto Project and Buildroot.

procps	Busybox	
us	usr	User space programs with default nice value
sy	sys	Kernel code
ni	nic	User space programs with non-default nice value
id	idle	Idle
wa	io	I/O wait
hi	irq	Hardware interrupts
si	sirq	Software interrupts
st	–	Steal time: only relevant in virtualized environments

Mem: 57044K used, 446172K free, 40K shrd, 3352K buff, 34452K cached
CPU: 58% usr 4% sys 0% nic 0% idle 37% io 0% irq 0% sirq
Load average: 0.24 0.06 0.02 2/51 105
PID PPID USER STAT VSZ %VSZ %CPU COMMAND
105 104 root R 27912 6% 61% ffmpeg -i track2.wav

Poor man's profler

1. Attach to the process using gdbserver (for a remote debug) or gbd (for a native debug). The process stops.
2. Observe the function it stopped in. You can use the backtrace GDB command to see the call stack.
3. Type continue so that the program resumes.
4. After a while, type Ctrl + C to stop it again and go back to step 2.

perf

perf is an abbreviation of the Linux performance event counter subsystem, perf_events, and also the name of the command-line tool for interacting with perf_events. Both have been part of the kernel since Linux 2.6.31. There is plenty of useful information in the Linux source tree in tools/perf/Documentation, and also at https://perf.wiki.kernel.org.

Confguring the kernel for perf

You need a kernel that is confgured for perf_events and you need the perf command cross compiled to run on the target. The relevant kernel confguration is CONFIG_PERF_EVENTS present in the menu General setup | Kernel Performance Events And Counters.

Building perf with the Yocto Project

EXTRA_IMAGE_FEATURES = "debug-tweaks dbg-pkgs
tools-profile"IMAGE_INSTALL_append = "kernel-vmlinux"

Building perf with Buildroot

- BR2_LINUX_KERNEL_TOOL_PERF in Kernel | Linux Kernel Tools. To build packages with debug symbols and install them unstripped on the target, select these two settings.
- BR2_ENABLE_DEBUG in the menu Build options | build packages with debugging symbols menu.
- BR2_STRIP = none in the menu Build options | strip command for binaries on target.

Profling with perf

```
perf record sh -c "find /usr/share | xargs grep linux > /dev/null"
perf record: Woken up 2 times to write data
perf record: Captured and wrote 0.368 MB perf.data (~16057 samples)
ls -l perf.data
-rw------- 1 root root 387360 Aug 25 2015 perf.data
```

perf user interfaces

- –stdio: This is a pure text interface with no user interaction. You will have to launch perf report and annotate for each view of the trace.
- –tui: This is a simple text-based menu interface with traversal between screens.
- –gtk: This is a graphical interface that otherwise acts in the same way as –tui.

```
Samples: 9K of event 'cycles', Event count (approx.): 2006177260
 11.29%   grep  libc-2.20.so       [.] re_search_internal
  8.80%   grep  busybox.nosuid     [.] bb_get_chunk_from_file
  5.55%   grep  libc-2.20.so       [.] _int_malloc
  5.40%   grep  libc-2.20.so       [.] _int_free
  3.74%   grep  libc-2.20.so       [.] realloc
  2.59%   grep  libc-2.20.so       [.] malloc
  2.51%   grep  libc-2.20.so       [.] regexec@@GLIBC_2.4
  1.64%   grep  busybox.nosuid     [.] grep_file
  1.57%   grep  libc-2.20.so       [.] malloc_consolidate
  1.33%   grep  libc-2.20.so       [.] strlen
  1.33%   grep  libc-2.20.so       [.] memset
  1.26%   grep  [kernel.kallsyms]  [k] __copy_to_user_std
  1.20%   grep  libc-2.20.so       [.] free
  1.10%   grep  libc-2.20.so       [.] _int_realloc
  0.95%   grep  libc-2.20.so       [.] re_string_reconstruct
  0.79%   grep  busybox.nosuid     [.] xrealloc
  0.75%   grep  [kernel.kallsyms]  [k] __do_softirq
  0.72%   grep  [kernel.kallsyms]  [k] preempt_count_sub
  0.68%   find  [kernel.kallsyms]  [k] __do_softirq
  0.53%   grep  [kernel.kallsyms]  [k] __dev_queue_xmit
  0.52%   grep  [kernel.kallsyms]  [k] preempt_count_add
  0.47%   grep  [kernel.kallsyms]  [k] finish_task_switch.isra.85
Press '?' for help on key bindings
```

OProfle and gprof

- CONFIG_PROFILING in General setup | Profiling support
- CONFIG_OPROFILE in General setup | OProfile system profiling

operf <program>
busybox grep łinux"*
ls -l gmon.out
-rw-r–r– 1 root root 473 Nov 24 14:07 gmon.out
gprof busybox
Flat profile:
Each sample counts as 0.01 seconds.
no time accumulated
% cumulative self self total time seconds seconds calls Ts/call Ts/call name
0.00 0.00 0.00 688 0.00 0.00 xrealloc
0.00 0.00 0.00 345 0.00 0.00 bb_get_chunk_from_file
0.00 0.00 0.00 345 0.00 0.00 xmalloc_fgetline
0.00 0.00 0.00 6 0.00 0.00 fclose_if_not_stdin
0.00 0.00 0.00 6 0.00 0.00 fopen_for_read
0.00 0.00 0.00 6 0.00 0.00 grep_file
..

LTTng components

- A core session manager
- A kernel tracer implemented as a group of kernel modules
- A user space tracer implemented as a library

LTTng and the Yocto Project

IMAGE_INSTALL_append = łttng-tools lttng-modules lttng-ust"

LTTng and Buildroot

- BR2_PACKAGE_LTTNG_MODULES in the menu Target packages | Debugging, profiling and benchmark | lttng-modules.
- BR2_PACKAGE_LTTNG_TOOLS in the menu Target packages | Debugging, profiling and benchmark | lttng-tools.
- BR2_PACKAGE_LTTNG_LIBUST in the menu Target packages | Libraries | Other, enable lttng-libust.

Callgrind

valgrind –tool=callgrind <program>

Helgrind

valgrind –tool=helgrind <program>

Using strace to show system calls

- Learn which system calls a program makes.
- Find those system calls that fail together with the error code. I find this useful if a program fails to start but doesn't print an error message or if the message is too general. strace shows the failing syscall.
- Find which files a program opens.
- Find out what syscalls a running program is making, for example to see if it is stuck in a loop.

```
strace ./helloworld
execve("./helloworld", ["./helloworld"], [/* 14 vars */]) = 0
brk(0) = 0x11000
uname({sys=Łinux", node="beaglebone", ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
1, 0) = 0xb6f40000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=8100, ...}) = 0
mmap2(NULL, 8100, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb6f3e000
close(3) = 0
```

References

C. Simmonds.
Mastering Embedded Linux Programming.
Packt Publishing, 2015.

The End